

`ucd_plot`

March 20, 2013
17:17

Contents

1 Create a SILO file for plotting using the polygon netCDF file	1
2 INDEX	20

1 Create a SILO file for plotting using the polygon netCDF file

An unstructured (ucd) mesh is created from the *g2* class. This program is analogous to and in many ways similar to *geomtesta*. The principal difference is that the area (or volume) “visualized” with *geomtesta* is specified on input; in *ucd_plot*, it is the entire problem, as described by the polygon netCDF file. As such, this program is only applicable to 2-D problems. The big advantage of *ucd_plot* over *geomtesta* is that the output data in the SILO file are effectively represented on the actual DEGAS2 grid, rather than an approximated, pixelated representation.

The SILO library is required to run *ucd_plot*; there is no HDF4 counterpart. Moreover, visualization of the resulting files has been performed only with the VisIt package (see <https://wci.llnl.gov/codes/visit/home.html>). The particular method used by *ucd_plot* to build the unstructured mesh relies on a simplified representation for the mesh that is not documented in the SILO library and may be peculiar to VisIt.

The command line invoking *ucd_plot* is:

```
ucd_plot poly.nc file_list
```

where *poly.nc* is the polygon netCDF file for the present geometry as written by *definegeometry2d* is a mandatory argument. Note that only *definegeometry2d* presently generates this file.

Due to a problem with the VTK renderer, convex polygons will not be handled correctly in VisIt. In particular, “pseudocolor” plots will have regions that appear to be masked by the concave polygons. Two workarounds are available:

- Use either *triangulate_polygon* or *triangulate_to_zones* in *definegeometry2d* (as opposed to *breakup_polygon*); all polygons will then be represented as triangles in the polygon netCDF file.
- If the only convex polygons in your geometry are solid zones, and they are the last zone(s) in the geometry, you can enable the section of code below labeled “Convex Polygon Hack”.

This is a known problem with VisIt / VTK and may be fixed at some point.

The output of the routine is a SILO format file in the same directory and with the same name as the run’s primary output netCDF file, but with a *.silo* extension instead of *.nc*.

The second argument is optional and contains a list of one or more text files with external data to be plotted. Each of these text files can have keywords “name”, “format” (as in a Fortran output specification; not used here, though), “units” to provide additional information on the data. If provided the name will be used to name the variable. If not, the name will be *ext_var_n* where *n* is the number of the file in the list of external files. All other lines in the file should be in the form:

```
zone    data
```

where *zone* is an integer between 1 and *zn_num* (inclusive) and **data** is the (real) data value. One way to generate such files within DEGAS 2 would be to use *outputbrowser* to write out zone based data (e.g., for a tally unique to the problem at hand) to a file and then edit the file to these specifications.

Here is an explicit example:

```
name neu_ob
units m^-3
  1    8.57659E+14
  2    6.70615E+14
  3    7.16331E+14
  4    6.29992E+14
  5    5.83773E+14
```

This technique can also be used for time dependent data. If one were looping through a sequence of time intervals, data from each step could be stored in text files of this format. In SILO mode, time dependent data should be stored in separate SILO files, one per time step. The convention used here is to append to the variable “name” an integer designation for the time step preceded by a \$ sign, e.g., **density_**\$00100 for the density at time step 100. The code will interpret the presence of the \$ sign as indicating time dependent data and use the following integer in naming the SILO file.

```
$Id: ucd_plot.web,v 1.1 2012/08/24 16:01:07 dstotler Exp $
"ucd_plot.f" 1≡
@m FILE 'ucd_plot.web'
```

The Main Program.

```
"ucd_plot.f" 1.1 ≡
program ucd_plot
implicit none_f77
implicit none_f90
integer nargs, poly_nc_fileid, ext_file_unit
character*FILELEN polygon_nc_file, ext_file_list
< Memory allocation interface 0 >
sy_decls
st_decls
nc_decls
call readfilenames
call degas_init
call nc_read_output
nargs = arg_count()
if (nargs < 1) then
    assert('Command_line must specify the name of the polygon netCDF file' ≡ ' ')
end if
call command_arg(1, polygon_nc_file)
poly_nc_fileid = ncopen(polygon_nc_file, NC_NOWRITE, nc_stat)
if (nc_stat ≠ 0) then
    assert('That polygon netCDF file cannot be opened!' ≡ ' ')
end if
if (nargs ≡ 2) then
    call command_arg(2, ext_file_list)
    ext_file_unit = diskin
    open(unit = ext_file_unit, file = ext_file_list, status = 'old', form = 'formatted')
else
    ext_file_unit = int_unused
end if
call make_plots(poly_nc_fileid, ext_file_unit)
stop
end
< Functions and Subroutines 1.2 >
```

Generate the SILO file and requisite data.

```
/* Remove characters causing file name problems */

"ucd_plot.f" 1.2 ≡
@m name_clean(s, sp) sp = s
    ind_tmp = index(sp, '(')
    if (ind_tmp > 0)
        sp(ind_tmp : ind_tmp) = '_'
    ind_tmp = index(sp, ')')
    if (ind_tmp > 0)
        sp(ind_tmp : ind_tmp) = '_'
    ind_tmp = index(sp, '|')
    if (ind_tmp > 0)
        sp(ind_tmp : ind_tmp) = '_'
    ind_tmp = index(sp, '+')
    if (ind_tmp > 0)
        sp(ind_tmp : ind_tmp) = 'p'

⟨Functions and Subroutines 1.2⟩ ≡
subroutine make_plots(poly_nc_fileid, ext_file_unit)

    define_dimen(ext_file_ind, n_ext_file)
    define_dimen(node_ind, num_nodes)
    define_dimen(poly_points_ind, num_poly_points)

    define_varp(polygon_data, FLOAT, g2_poly_ind)
    define_varp(zone_data, FLOAT, zone_ind)
    define_varp(ext_filenames, CHAR, filenames_size, ext_file_ind)
    define_varp(nodes, FLOAT, g2_xz_ind, node_ind)
    define_varp(node_x, FLOAT, node_ind)
    define_varp(node_z, FLOAT, node_ind)
    define_varp(boundary_nodes, INT, node_ind)
    define_varp(poly_pointlist, INT, poly_points_ind)

    implicit none_f77
    sp_common
    pr_common
    tl_common
    ou_common
    bk_common
    zn_common
    de_common
    rf_common
    sc_common
    g2_common
    implicit none_f90

    integer nargs, poly_nc_fileid, ipoly, zone, num_nodes, num_points, mesh_sense, i, j, this_node,
           sector, num_poly_points, i_this, dbfile, ret, ret2, nshapetypes, ndims, n_ext_file, ext_file_unit,
           length, beg, e, p, iview, test, back, ifile, itxt, open_stat, ind_tmp, iv, last_plasma_polygon,
           dbfile_td

    integer shapesize_1, shapecounts_1, shapetype_1, this_polygon_0:g2_num_points-1,
           index_parameters tl_index_max
    logical dbfile_td_open
```

```

real max_v, ha_temp, density, ha_rate, pressure, halpha_tot
character*FILELEN polygon_nc_file, silo_file, zonelist, mesh_name, ext_silo_file
character*4 ivlab
character*1 ind_sy3
character*3 iflab
character*3 vtag, auth
character*tl_tag_length vname
character*sp_sy_len clean_sy
character*LINELEN line, ext_var_name, ext_var_units, ext_var_format
data ind_sy/’1’, ’2’, ’3’/

external extract_output_datum
real extract_output_datum

vc_decl(yhat)
vc_decl(test_vec_1)
vc_decl(test_vec_2)
vc_decl(test_vec_3)
vc_decl(node_vec)

declare_varp(polygon_data)
declare_varp(zone_data)
declare_varp(ext_filenames)
declare_varp(nodes)
declare_varp(node_x)
declare_varp(node_z)
declare_varp(boundary_nodes)
declare_varp(poly_pointlist)

⟨ Memory allocation interface 0 ⟩
sy_decls
vc_decls
g2_ncdecl
nc_decls
st_decls
tl_decls
zn_decls

include ‘silo.inc’

e = index(filenames_array outputfile, ‘.nc’) - 1
assert(e > 0)
silo_file = filenames_array outputfile SP(: e) || ‘_ucd.silo’
ret = dbcreate(trim(silo_file), string_length(silo_file), DB_CLOBBER, DB_LOCAL, DB_F77NULL,
               0, silo_format, dbfile)
assert((ret ≠ -1) ∧ (dbfile ≠ -1))
dbfile_td_open = F

num_nodes = 0
num_poly_points = 0
var_alloc(nodes)
var_alloc(boundary_nodes)
var_alloc(poly_pointlist)
var_alloc(zone_data)
/* Check the end of the file for additional (external) zone based data in text files. */
n_ext_file = 0

```

```

var_alloc(ext_filenames)
if (ext_file_unit ≠ int_unused) then
loop: continue
  if (read_string(ext_file_unit, line, length)) then
    assert(length ≤ len(line))
    length = parse_string(line(:length))
    p = 0
    assert(next_token(line, beg, e, p))
    n_ext_file++
    var_realloc(ext_filenames)
    ext_filenamesn_ext_file = line(beg : e)
    go to loop
  end if
  close(unit = ext_file_unit)
end if /* Set up and write unstructured mesh to SILO file. */
g2_ncread(poly_nc_fileid) /* Convex Polygon Hack As described in the preamble, if the only
convex polygons in your geometry are solid zones, and they are the last zone(s) in the geometry,
you can enable this section of code by changing if 0 to if 1 and recompiling the code. This
will skip the solid zones when creating the ucd mesh. */
@if 1
last_plasma_polygon = 0
do ipoly = 1, g2_num_polygons
  zone = g2_polygon_zoneipoly
  if (last_plasma_polygon ≡ 0) then
    if ((zn_type(zone) ≠ zn_plasma) ∧ (zn_type(zone) ≠ zn_vacuum)) then
      last_plasma_polygon = ipoly - 1
      assert(last_plasma_polygon > 0)
    end if
  else
    assert((zn_type(zone) ≠ zn_plasma) ∧ (zn_type(zone) ≠ zn_vacuum))
  end if
end do
g2_num_polygons = last_plasma_polygon // Reset for convenience
#endif
var_alloc(polygon_data)
vc_set(yhat, zero, one, zero)
do ipoly = 1, g2_num_polygons
  /* Use the first two segments to determine the orientation of the zone. */
  vc_set(test_vec_1, g2_polygon_xzipoly,1,g2_x - g2_polygon_xzipoly,0,g2_x, zero,
         g2_polygon_xzipoly,1,g2_z - g2_polygon_xzipoly,0,g2_z)
  vc_set(test_vec_2, g2_polygon_xzipoly,2,g2_x - g2_polygon_xzipoly,1,g2_x, zero,
         g2_polygon_xzipoly,2,g2_z - g2_polygon_xzipoly,1,g2_z)
  vc_cross(test_vec_1, test_vec_2, test_vec_3)
  if (vc_product(test_vec_3, yhat) > zero) then
    mesh_sense = 1 // clockwise
  else if (vc_product(test_vec_3, yhat) < zero) then
    mesh_sense = 2 // counter-clockwise
  else
    assert('Mesh_cell_degenerate' ≡ ' ')
  end if /* One the tricks used in setting up meshes for XGC0 is to encode triangles into the
Sonnet mesh by making the last two points the same. Reduce the number of points in the
polygon by one if they are the same. */

```

```

num_points = g2_polygon_points(ipoly)
if ((g2_polygon_xz_ipoly,num_points-1,g2_x == g2_polygon_xz_ipoly,num_points,g2_x)  $\wedge$ 
     (g2_polygon_xz_ipoly,num_points-1,g2_z == g2_polygon_xz_ipoly,num_points,g2_z))
  num_points = num_points - 1 /* The convention in definegeometry2d is that the zero-th and
                                n-th point are the same. Hence, the range for this loop. */
do i = 0, num_points - 1
  this_node = 0
  if (num_nodes > 0) then
    do j = 1, num_nodes
      if ((nodesj,g2_x == g2_polygon_xzipoly,i,g2_x)  $\wedge$  (nodesj,g2_z == g2_polygon_xzipoly,i,g2_z)) then
        this_node = j
      end if
    end do
  end if
  if (this_node == 0) then
    num_nodes++
    var_realloc(nodes)
    var_realloc(boundary_nodes)
    nodesnum_nodes,g2_x = g2_polygon_xzipoly,i,g2_x
    nodesnum_nodes,g2_z = g2_polygon_xzipoly,i,g2_z
    /* Determine whether or not this node lies on a "boundary". Since we have access to all of
       the geometry data, this amounts to searching through the wall, target, and exit sectors to
       see if it matches one of their points. */
    vc_set(node_vec, nodesnum_nodes,g2_x, zero, nodesnum_nodes,g2_z)
    boundary_nodesnum_nodes = 0 // Not on boundary
    do sector = 1, nsectors
      if (((sc_target_check(sector_type_pointersector,sc_target))  $\vee$ 
            (sc_wall_check(sector_type_pointersector,sc_wall))  $\vee$ 
            (sc_exit_check(sector_type_pointersector,sc_exit)))  $\wedge$  ((vc_equal(node_vec,
              sector_pointssector,sc_neg)  $\vee$  (vc_equal(node_vec, sector_pointssector,sc_pos)))) then
        boundary_nodesnum_nodes = 1 // Is on boundary
      end if
    end do
    this_node = num_nodes
  end if
  this_polygoni = this_node
end do
// Over i /* Add the nodes for this_polygon to the global list of points for all polygons. Here
we make use of SILO's UCD mesh feature in which we tell it that the polygons all have zero
points and use the first entry in the points list to contain the number of points, hence the +1.
num_poly_points++
var_realloc(poly_pointlist)
poly_pointlistnum_poly_points = num_points
do i = 0, num_points - 1
  num_poly_points++
  var_realloc(poly_pointlist)
  if (mesh_sense == 1) then
    i_this = num_points - 1 - i // Reverse clockwise polygons
  else
    i_this = i
  end if
  poly_pointlistnum_poly_points = this_polygoni_this

```

```

    end do
end do

nshapetypes = 1
ndims = 2
shapesize_1 = 0
shapecounts_1 = g2_num_polygons
shapetype_1 = DB_ZONETYPE_POLYGON
zonelist = 'polygon_zonelist'

ret = dbputzl2(dbfile, trim(zonelist), string_length(zonelist), g2_num_polygons, ndims, poly_pointlist,
               num_poly_points, 1, 0, 0, shapetype, shapesize, shapecounts, nshapetypes, DB_F77NULL, ret2)
assert((ret != -1) & (ret2 != -1))
mesh_name = 'polygon_mesh' /* SILO expects the x, y, and z coordinates of the nodes in
                           separate arrays. For the moment at least, define these now from nodes. Could later replace it in
                           all of the above with node_x and node_z. */

var_alloc(node_x)
var_alloc(node_z)
do i = 1, num_nodes
  node_x_i = nodes_i,g2_x
  node_z_i = nodes_i,g2_z
end do /* This is presently hardcoded for 2-D; may be able to do 3-D eventually. */
ret = dbputum(dbfile, trim(mesh_name), string_length(mesh_name), ndims, node_x, node_z,
              DB_F77NULL, "x", 1, "z", 1, DB_F77NULLSTRING, 0, silo_precision, num_nodes,
              g2_num_polygons, trim(zonelist), string_length(zonelist), DB_F77NULLSTRING, 0,
              DB_F77NULL, ret2)
assert((ret != -1) & (ret2 != -1)) /* START WRITING DATA. PRESENTLY CRIBBED FROM
                                      GEOMTESTA. This is just Charles' original "zone function". I believe the purpose was to use
                                      (most of) an 8-bit palette to color the zones with a minimum chance of color matching between
                                      adjacent zones (i.e., suspect that 97/252 is related to the golden mean). */
do i = 1, g2_num_polygons
  zone = g2_polygon_zone_i
  polygon_data_i = areal(3 + mod(97 * zone, 255 - 3))
end do
call write_silo_data('zone_function', polygon_data, 'l', 'E11.3', dbfile, mesh_name)

do i = 1, g2_num_polygons
  zone = g2_polygon_zone_i
  polygon_data_i = areal(zone)
end do
call write_silo_data('zone_number', polygon_data, 'l', 'E11.3', dbfile, mesh_name)

do i = 1, g2_num_polygons
  zone = g2_polygon_zone_i
  polygon_data_i = zn_volume(zone)
end do
call write_silo_data('zone_volume', polygon_data, 'm**3', 'E11.3', dbfile, mesh_name)

do i = 1, g2_num_polygons
  zone = g2_polygon_zone_i
  polygon_data_i = areal(zn_type(zone))
end do
call write_silo_data('zone_type', polygon_data, 'l', 'E11.3', dbfile, mesh_name)

@if 1
do i = 1, g2_num_polygons

```

```

zone = g2_polygon_zone_i
if (zn_type(zone) ≡ zn_plasma) then
    polygon_datai = bk_n(1, zone) // BACKGROUND DENSITY
else
    polygon_datai = zero
end if
end do
call write_silo_data('electron_density', polygon_data, 'm**-3', 'E11.3', dbfile, mesh_name)

do i = 1, g2_num_polygons
    zone = g2_polygon_zone_i
    if (zn_type(zone) ≡ zn_plasma) then // BACKGROUND TEMPERATURE
        polygon_datai = bk_temp(1, zone) / electron_charge
    else
        polygon_datai = zero
    end if
end do
call write_silo_data('electron_temperature', polygon_data, 'eV', 'E11.3', dbfile, mesh_name)

do i = 1, g2_num_polygons
    zone = g2_polygon_zone_i
    if (zn_type(zone) ≡ zn_plasma) then // BACKGROUND TEMPERATURE
        polygon_datai = bk_temp(2, zone) / electron_charge
    else
        polygon_datai = zero
    end if
end do
call write_silo_data('ion_temperature', polygon_data, 'eV', 'E11.3', dbfile, mesh_name)

do j = 1, 3
    max_v = zero
    do i = 1, g2_num_polygons
        zone = g2_polygon_zone_i
        if (zn_type(zone) ≡ zn_plasma) then // ION VELOCITY
            polygon_datai = bk_v(2, zone)j
            max_v = max(max_v, abs(polygon_datai))
        else
            polygon_datai = zero
        end if
    end do
    if (max_v > zero) then
        call write_silo_data('ion_velocity_' || ind_syj, polygon_data, 'm/s', 'E11.3', dbfile,
                           mesh_name)
    end if
end do
@#endif
@#if 0 // Plots of de_zone frags

do j = 1, 4
    if (j ≡ 1) then
        iview = 1
        ivlab = '1'
    else if (j ≡ 2) then
        iview = 4
    end if

```

```

    ivlab = '4'
else if (j == 3) then
    iview = 6
    ivlab = '6'
else
    iview = 9
    ivlab = '9'
end if
do i = 1, g2_num_polygons
    zone = g2_polygon_zone_i
    if (zn_type(zone) == zn_plasma || zn_type(zone) == zn_vacuum) then
        polygon_data_i = de_zone frags zone,iview
    else
        polygon_data_i = zero
    end if
end do
call write_silo_data('zone_frag_view_' || trim(ivlab), polygon_data, 'L', 'E11.3', dbfile,
                     mesh_name)
end do
@ifendif
@if 1
assert(output_old_file == TRUE)
do j = 2, pr_test_num
    index_parameters tl_index_test = j
    ha_temp = zero
    do i = 1, g2_num_polygons
        zone = g2_polygon_zone_i
        if (zn_type(zone) == zn_plasma || zn_type(zone) == zn_vacuum) then
            index_parameters tl_index_zone = zone
            density = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
                                             'neutral_Ldensity')
            polygon_data_i = density // NEUTRAL DENSITY
            ha_temp += density * zn_volume(zone)
        else
            polygon_data_i = zero
        end if
    end do
    name_clean(sp_sy(pr_test(j)), clean_sy)
    call write_silo_data('sp' || trim(clean_sy) || '_density', polygon_data, 'm**-3', 'E11.3',
                         dbfile, mesh_name)

    do i = 1, g2_num_polygons
        zone = g2_polygon_zone_i
        if (zn_type(zone) == zn_plasma || zn_type(zone) == zn_vacuum) then
            index_parameters tl_index_zone = zone
            density = extract_output_datum(index_parameters, 1, output_all, o_var, 'neutral_Ldensity')
            polygon_data_i = density // NEUTRAL DENSITY REL. STD. DEV.
        else
            polygon_data_i = zero
        end if
    end do
    call write_silo_data('sp' || trim(clean_sy) || '_density_rsd', polygon_data, 'L', 'E11.3',
                         dbfile, mesh_name)
end if

```

```

        dbfile, mesh_name)

if (tl_check(string_lookup('neutral_velocity_vector', tally_name, tl_num))) then
    vname = 'neutral_velocity_vector'
    vtag = 'vel'
else
    assert(tl_check(string_lookup('neutral_flux_vector', tally_name, tl_num)))
    vname = 'neutral_flux_vector'
    vtag = 'flx'
end if
do iv = 1, 3 // Component
    do i = 1, g2_num_polygons
        zone = g2_polygon_zone_i
        if (zn_type(zone) ≡ zn_plasma ∨ zn_type(zone) ≡ zn_vacuum) then
            index_parameterstl_index_zone = zone
            ha_rate = extract_output_datum(index_parameters, iv, out_post_all, o_mean, trim(vname))
            polygon_datai = ha_rate
        else
            polygon_datai = zero
        end if
    end do
    call write_silo_data(trim(clean_sy) || vtag || ind_syiv, polygon_data, 'm/s', 'E11.3', dbfile,
        mesh_name)
end do

if (string_lookup('neutral_pressure', tally_name, tl_num) > 0) then
    do i = 1, g2_num_polygons
        zone = g2_polygon_zone_i
        if (zn_type(zone) ≡ zn_plasma ∨ zn_type(zone) ≡ zn_vacuum) then
            index_parameterstl_index_zone = zone
            pressure = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
                'neutral_pressure')
            polygon_datai = pressure / const(1.3332, -1) // NEUTRAL PRESSURE
        else
            polygon_datai = zero
        end if
    end do
    call write_silo_data('sp' || trim(clean_sy) || '_pressure', polygon_data, 'mTorr', 'E11.3',
        dbfile, mesh_name)
end if

end do
#endif
#if 0
/* This is provisional. Could just test for the presence of the tally and leave this in here
permanently. */
j = 2 // Usually the atom species
index_parameterstl_index_test = j
do k = 1, so_type_num + pr_reaction_num + pr_pmi_num
    index_parameterstl_index_test_author = k
    ha_temp = zero
    do i = 1, g2_num_polygons
        zone = g2_polygon_zone_i
        if (zn_type(zone) ≡ zn_plasma ∨ zn_type(zone) ≡ zn_vacuum) then

```

```

index_parameters tl_index_zone = zone
density = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
                               'neutral_density_by_author')
polygon_data_i = density // NEUTRAL DENSITY BY AUTHOR
ha_temp += density * zn_volume(zone)
else
  polygon_data_i = zero
end if
end do
if (ha_temp > zero) then
  write(auth, '(a1,i2.2)' 'a', k
  call write_silo_data('sp' || trim(sp_sy(pr_test(j))) || '_density' || auth, polygon_data,
                       'm**-3', 'E11.3', dbfile, mesh_name)
end if
end do

@ifendif
@if 1
do i = 1, g2_num_polygons
  polygon_data_i = zero
end do

do test = 1, pr_test_num
  if (((sp_sy(pr_test(test)) == 'H') /* NEUTRAL H-ALPHA */
    || (sp_sy(pr_test(test)) == 'D') || (sp_sy(pr_test(test)) ==
      'T')) & string_lookup(trim(sp_sy(pr_test(test))) || 'alpha_emission_rate', tally_name,
      tl_num) > 0) then
    ha_temp = zero
    do i = 1, g2_num_polygons
      zone = g2_polygon_zone_i
      if (zn_type(zone) == zn_plasma) then
        index_parameters tl_index_zone = zone
        ha_rate = extract_output_datum(index_parameters,
                                       1, out_post_all, o_mean, trim(sp_sy(pr_test(test))) ||
                                       'alpha_emission_rate') / (const(1.8881944) * electron_charge)
        polygon_data_i += ha_rate
        ha_temp += ha_rate * zn_volume(zone)
      else if (zn_type(zone) == zn_solid) then // Can plot solid
        polygon_data_i = -one // regions
      endif
    end do
  end if
end do

halpha_tot = zero
do i = 1, g2_num_polygons
  zone = g2_polygon_zone_i
  halpha_tot += polygon_data_i * zn_volume(zone)
end do

if (halpha_tot > zero) then

```

```

call write_silo_data('H_alpha_rate', polygon_data, 'photons'/(m**3*s), 'E11.3', dbfile,
                     mesh_name)
end if

do i = 1, g2_num_polygons
  polygon_data_i = zero
end do

do test = 1, pr_test_num
  if (string_lookup('Dalpha_emission_rate_by_species', tally_name, tl_num) > 0) then
    name_clean(sp_sy(pr_test(test)), clean_sy)
    ha_temp = zero
    index_parameters tl_index_test = test
    do i = 1, g2_num_polygons
      zone = g2_polygon_zone_i
      if (zn_type(zone) == zn_plasma) then
        index_parameters tl_index_zone = zone
        ha_rate = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
                                        'Dalpha_emission_rate_by_species') / (const(1.8881944) * electron_charge)
        polygon_data_i = ha_rate
        ha_temp += ha_rate * zn_volume(zone)
      else
        polygon_data_i = zero
      end if
    end do
  end if
  halpha_tot = zero
  do i = 1, g2_num_polygons
    zone = g2_polygon_zone_i
    halpha_tot += polygon_data_i * zn_volume(zone)
  end do

  if (halpha_tot > zero) then
    call write_silo_data('H_alpha_rate_ ' || trim(clean_sy), polygon_data,
                         'photons'/(m**3*s), 'E11.3', dbfile, mesh_name)
  end if
end do

#endif
#ifndef 1
do i = 1, g2_num_polygons
  polygon_data_i = zero
end do

do test = 1, pr_test_num
  if (((sp_sy(pr_test(test)) == 'He') /* NEUTRAL He 5877 line */ 
       & string_lookup('He_5877_emission_rate', tally_name, tl_num) > 0)) then
    ha_temp = zero
    do i = 1, g2_num_polygons
      zone = g2_polygon_zone_i
      if (zn_type(zone) == zn_plasma) then
        index_parameters tl_index_zone = zone
        ha_rate = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
                                       'He_5877_emission_rate') / (const(2.109566) * electron_charge)
        polygon_data_i += ha_rate
        ha_temp += ha_rate * zn_volume(zone)
      end if
    end do
  end if
end do

```

```

@#if 0
else if (zn_type(zone) == zn_solid) then // Can plot solid
    polygon_data_i = -one // regions
#endif
else
    polygon_data_i = zero
end if
end do
end if
end do

halpha_tot = zero
do i = 1, g2_num_polygons
    zone = g2_polygon_zone_i
    halpha_tot += polygon_data_i * zn_volume(zone)
end do

if (halpha_tot > zero) then
    call write_silo_data('He_5877_rate', polygon_data, 'photons'/(m**3*s), 'E11.3', dbfile,
                         mesh_name)
end if
#endif

if ((pr_reaction_num > 0) ∧ (string_lookup('ion_source_rate', tally_name, tl_num) > 0)) then
    do back = 1, pr_background_num
        name_clean(sp_sy(pr_background(back)), clean_sy)
        if (trim(clean_sy) ≠ 'e') then // ION SOURCE RATE
            index_parameters tl_index_problem_sp = pr_problem_sp_back(back)
            ha_temp = zero
            do i = 1, g2_num_polygons
                zone = g2_polygon_zone_i
                if (zn_type(zone) == zn_plasma) then
                    index_parameters tl_index_zone = zone
                    ha_rate = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
                        'ion_source_rate')
                    polygon_data_i = ha_rate / zn_volume(zone)
                    ha_temp += ha_rate
                else
                    polygon_data_i = zero
                end if
            end do
            call write_silo_data(trim(clean_sy) || '_Ion_Source_rate', polygon_data, 'm**-3*s**-1',
                                'E11.3', dbfile, mesh_name)
            do i = 1, g2_num_polygons
                zone = g2_polygon_zone_i
                if (zn_type(zone) == zn_plasma) then
                    index_parameters tl_index_zone = zone
                    ha_rate = extract_output_datum(index_parameters, 1, output_all, o_var,
                        'ion_source_rate')
                    polygon_data_i = ha_rate
                else
                    polygon_data_i = zero
                end if
            end do
        end if
    end do
end if

```

```

call write_silo_data(trim(clean_sy) || '_Ion_Source_rate_rsd', polygon_data, 'u',
                     'E11.3', dbfile, mesh_name)

end if
end do

do back = 1, pr_background_num
  name_clean(sp_sy(pr_background(back)), clean_sy)
  if(trim(clean_sy) .neq. 'e') then // MOM. SOURCE RATE
    index_parameters tl_index_problem_sp = pr_problem_sp_back(back)
    do iv = 1, 3 // Component
      ha_temp = zero
      do i = 1, g2_num_polygons
        zone = g2_polygon_zone_i
        if(zn_type(zone) .equiv. zn_plasma) then
          index_parameters tl_index_zone = zone
          ha_rate = extract_output_datum(index_parameters, iv, out_post_all, o_mean,
                                         'ion_momentum_source_vector')
          polygon_data_i = ha_rate / zn_volume(zone)
          ha_temp += ha_rate
        else
          polygon_data_i = zero
        end if
      end do
      call write_silo_data(trim(clean_sy) || '_Mom_' || ind_sy_iv || '_Source_rate',
                           polygon_data, 'Nmm**-3', 'E11.3', dbfile, mesh_name)
    end do
    do iv = 1, 3 // Component
      do i = 1, g2_num_polygons
        zone = g2_polygon_zone_i
        if(zn_type(zone) .equiv. zn_plasma) then
          index_parameters tl_index_zone = zone
          ha_rate = extract_output_datum(index_parameters, iv, output_all, o_var,
                                         'ion_momentum_source_vector')
          polygon_data_i = ha_rate
        else
          polygon_data_i = zero
        end if
      end do
      call write_silo_data(trim(clean_sy) || '_Mom_' || ind_sy_iv || '_Source_rate_rsd',
                           polygon_data, 'u', 'E11.3', dbfile, mesh_name)
    end do
  end if
end do

do back = 1, pr_background_num // ENERGY SOURCE RATE
  name_clean(sp_sy(pr_background(back)), clean_sy)
  index_parameters tl_index_problem_sp = pr_problem_sp_back(back)
  ha_temp = zero
  do i = 1, g2_num_polygons
    zone = g2_polygon_zone_i
    if(zn_type(zone) .equiv. zn_plasma) then
      index_parameters tl_index_zone = zone

```

```

ha_rate = extract_output_datum(index_parameters, 1, out_post_all, o_mean,
    'ion_energy_source')
polygon_data_i = ha_rate / zn_volume(zone)
ha_temp += ha_rate
else
    polygon_data_i = zero
end if
end do
call write_silo_data(trim(clean_sy) || '_Energy_Source_rate', polygon_data, 'W_m**-3',
    'E11.3', dbfile, mesh_name)

do i = 1, g2_num_polygons
    zone = g2_polygon_zone_i
    if (zn_type(zone) ≡ zn_plasma) then
        index_parameters_tl_index_zone = zone
        ha_rate = extract_output_datum(index_parameters, 1, output_all, o_var,
            'ion_energy_source')
        polygon_data_i = ha_rate
    else
        polygon_data_i = zero
    end if
end do
call write_silo_data(trim(clean_sy) || '_Energy_Source_rate_rsd', polygon_data, 'u',
    'E11.3', dbfile, mesh_name)
end do

end if /* Read in and plot external zone-based data. */
if (n_ext_file > 0) then
    do ifile = 1, n_ext_file
        do i = 1, zn_num
            zone_data_i = zero
        end do
        open(unit = diskin, file = ext_filenames_ifile, status = 'old', form = 'formatted',
            iostat = open_stat)
        if (open_stat ≡ 0) then
            assert(ifile < 1000)
            write(iflab, '(i3.3)') ifile
            ext_var_name = 'ext_var' || iflab
            ext_var_units = 'u'
            ext_var_format = 'E11.3'
        loop2: continue
        if (read_string(diskin, line, length)) then
            assert(length ≤ len(line))
            length = parse_string(line(:length))
            p = 0
            assert(next_token(line, beg, e, p))
            if (line(beg : e) ≡ 'name') then
                assert(next_token(line, beg, e, p))
                ext_var_name = line(beg : e)
            else if (line(beg : e) ≡ 'units') then
                assert(next_token(line, beg, e, p))
                ext_var_units = line(beg : e)
            else if (line(beg : e) ≡ 'format') then

```

```

    assert(next_token(line, beg, e, p))
    ext_var_format = line(beg : e)
else
    zone = read_int_soft_fail(line(beg : e))
    if (zn_check(zone)) then
        assert(next_token(line, beg, e, p))
        zone_data_zone = read_real(line(beg : e))
    else
        write(stderr, *) 'Improper zone number', zone, 'in external file',
        ext_filenames_n_ext_file
    end if
end if
go to loop2
end if
close(unit = diskin)
do i = 1, g2_num_polygons
    zone = g2_polygon_zone_i
    polygon_data_i = zone_data_zone
end do
i = index(ext_var_name, '$')
if (i > 0) then
    if (dbfile_td_open) then
        ret = dbclose(dbfile_td)
        dbfile_td_open = F // Not sure if I need this
        assert(ret ≠ -1)
    end if
    itxt = index(ext_filenames_ifile, '.txt')
    ext_silo_file = ext_filenames_ifile SP(: itxt) || 'silo'
    ret = dbcreate(trim(ext_silo_file), string_length(ext_silo_file), DB_CLOBBER, DB_LOCAL,
                  DB_F77NULL, 0, silo_format, dbfile_td)
    assert((ret ≠ -1) ∧ (dbfile_td ≠ -1))
    dbfile_td_open = T /* Write zone list and mesh to this file. */
    ret = dbputzl2(dbfile_td, trim(zonelist), string_length(zonelist), g2_num_polygons, ndims,
                  poly_pointlist, num_poly_points, 1, 0, 0, shapetype, shapesize, shapecounts,
                  nshapetypes, DB_F77NULL, ret2)
    assert((ret ≠ -1) ∧ (ret2 ≠ -1))
    ret = dbputum(dbfile_td, trim(mesh_name), string_length(mesh_name), ndims, node_x,
                  node_z, DB_F77NULL, "x", 1, "z", 1, DB_F77NULLSTRING, 0, silo_precision,
                  num_nodes, g2_num_polygons, trim(zonelist), string_length(zonelist),
                  DB_F77NULLSTRING, 0, DB_F77NULL, ret2)
    assert((ret ≠ -1) ∧ (ret2 ≠ -1))
    call write_silo_data(trim(ext_var_name), polygon_data, trim(ext_var_units),
                         trim(ext_var_format), dbfile_td, mesh_name)
else
    call write_silo_data(trim(ext_var_name), polygon_data, trim(ext_var_units),
                         trim(ext_var_format), dbfile, mesh_name)
end if
else
    write(stderr, *) 'Cannot open external file', ext_filenames_n_ext_file,
    ', error number', open_stat
end if
end do

```

```
end if

ret = dbclose(dbfile)
assert(ret != -1)
if (dbfile_td_open) then
    ret = dbclose(dbfile_td)
    assert(ret != -1)
end if
var_free(nodes)
var_free(node_x)
var_free(node_z)
var_free(boundary_nodes)
var_free(poly_pointlist)
var_free(polygon_data)
var_free(zone_data)

return
end
```

See also section 1.3.

This code is used in section 1.1.

Write data to the SILO file.

```

⟨ Functions and Subroutines 1.2 ⟩ +≡
  subroutine write_silo_data(var_name, polygon_data, cunits, cformt, dbfile, mesh_name)
    implicit none_f77
    g2_common
    implicit none_f90

    integer dbfile      // Input
    character*(*) var_name, cunits, cformt, mesh_name
    real polygon_data_g2_num_polygons

    real time_slice
    integer ret, ret2, var_opts, time_ind

    st_decls
    include 'silo.inc'

    ret = dbmkoptlist(1, var_opts)
    assert(ret ≠ -1)
    if (len(cunits) > 0) then
        ret = dbaddcopt(var_opts, DBOPT_UNITS, cunits, len(cunits))
        assert(ret ≠ -1)
    end if
    time_ind = index(var_name, '$')
    if (time_ind > 0) then
        time_slice = areal(read_integer(var_name(time_ind + 1 :)))
        ret = dbadddopt(var_opts, DBOPT_DTIME, time_slice)
        assert(ret ≠ -1)
        var_name = var_name(: time_ind - 1)
    end if
    ret = dbputuv1(dbfile, trim(var_name), string_length(var_name), trim(mesh_name),
                  string_length(mesh_name), polygon_data, g2_num_polygons, DB_F77NULL, 0, silo_precision,
                  DB_ZONECENT, var_opts, ret2)
    assert((ret ≠ -1) ∧ (ret2 ≠ -1))

    return
end

```

2 INDEX

abs: 1.2.
 areal: 1.2, 1.3.
arg_count: 1.1.
assert: 1.1, 1.2, 1.3.
auth: 1.2.
back: 1.2.
beg: 1.2.
bk_common: 1.2.
bk_n: 1.2.
bk_temp: 1.2.
bk_v: 1.2.
boundary_nodes: 1.2.
breakup_polygon: 1.
cformat: 1.3.
CHAR: 1.2.
clean_sy: 1.2.
command_arg: 1.1.
const: 1.2.
cunits: 1.3.
DB_CLOBBER: 1.2.
DB_F77NULL: 1.2, 1.3.
DB_F77NULLSTRING: 1.2.
DB_LOCAL: 1.2.
DB_ZONECENT: 1.3.
DB_ZONETYPE_POLYGON: 1.2.
dbaddcpt: 1.3.
dbadddopt: 1.3.
dbclose: 1.2.
dbcreate: 1.2.
dbfile: 1.2, 1.3.
dbfile_td: 1.2.
dbfile_td_open: 1.2.
dbmkoptlist: 1.3.
DBOPT_DTIME: 1.3.
DBOPT_UNITS: 1.3.
dbputum: 1.2.
dbputuv1: 1.3.
dbputzl2: 1.2.
de_common: 1.2.
de_zone_frags: 1.2.
declare_varp: 1.2.
define_dimen: 1.2.
define_varp: 1.2.
definegeometry2d: 1, 1.2.
degas_init: 1.1.
density: 1.2.
diskin: 1.1, 1.2.
e: 1.2.
electron_charge: 1.2.
ext_file_ind: 1.2.
ext_file_list: 1.1.
ext_file_unit: 1.1, 1.2.
ext_filenames: 1.2.
ext_silo_file: 1.2.
ext_var_format: 1.2.
ext_var_n: 1.
ext_var_name: 1.2.
ext_var_units: 1.2.
extract_output_datum: 1.2.
file: 1.1, 1.2.
FILE: 1.
FILELEN: 1.1, 1.2.
filenames_array: 1.2.
filenames_size: 1.2.
FLOAT: 1.2.
form: 1.1, 1.2.
geomtesta: 1.
g2: 1.
g2_common: 1.2, 1.3.
g2_ncdecl: 1.2.
g2_ncread: 1.2.
g2_num_points: 1.2.
g2_num_polygons: 1.2, 1.3.
g2_poly_ind: 1.2.
g2_polygon_points: 1.2.
g2_polygon_xz: 1.2.
g2_polygon_zone: 1.2.
g2_x: 1.2.
g2_xz_ind: 1.2.
g2_z: 1.2.
ha_rate: 1.2.
ha_temp: 1.2.
halpha_tot: 1.2.
i: 1.2.
i_this: 1.2.
ifile: 1.2.
iflab: 1.2.
implicit_none_f77: 1.1, 1.2, 1.3.
implicit_none_f90: 1.1, 1.2, 1.3.
include: 1.2, 1.3.
ind_sy: 1.2.
ind_tmp: 1.2.
index: 1.2, 1.3.
index_parameters: 1.2.
INT: 1.2.
int_unused: 1.1, 1.2.
iostat: 1.2.

ipoly: 1.2.
itxt: 1.2.
iv: 1.2.
iview: 1.2.
ivlab: 1.2.
j: 1.2.
last_plasma_polygon: 1.2.
len: 1.2, 1.3.
length: 1.2.
line: 1.2.
LINELEN: 1.2.
loop: 1.2.
loop2: 1.2.
make_plots: 1.1, 1.2.
max: 1.2.
max_v: 1.2.
mesh_name: 1.2, 1.3.
mesh_sense: 1.2.
mod: 1.2.
n_ext_file: 1.2.
name_clean: 1.2.
nargs: 1.1, 1.2.
nc: 1.
nc_decls: 1.1, 1.2.
NC_NOWRITE: 1.1.
nc_read_output: 1.1.
nc_stat: 1.1.
ncopen: 1.1.
ndims: 1.2.
next_token: 1.2.
node_ind: 1.2.
node_vec: 1.2.
node_x: 1.2.
node_z: 1.2.
nodes: 1.2.
nsectors: 1.2.
nshapetypes: 1.2.
num_nodes: 1.2.
num_points: 1.2.
num_poly_points: 1.2.
o_mean: 1.2.
o_var: 1.2.
one: 1.2.
open_stat: 1.2.
ou_common: 1.2.
out_post_all: 1.2.
output_all: 1.2.
output_old_file: 1.2.
outputbrowser: 1.
outputfile: 1.2.
p: 1.2.
parse_string: 1.2.
poly: 1.
poly_nc_fileid: 1.1, 1.2.
poly_pointlist: 1.2.
poly_points.ind: 1.2.
polygon_data: 1.2, 1.3.
polygon_nc_file: 1.1, 1.2.
pr_background: 1.2.
pr_background_num: 1.2.
pr_common: 1.2.
pr_pmi_num: 1.2.
pr_problem_sp_back: 1.2.
pr_reaction_num: 1.2.
pr_test: 1.2.
pr_test_num: 1.2.
pressure: 1.2.
read_int_soft_fail: 1.2.
read_integer: 1.3.
read_real: 1.2.
read_string: 1.2.
readfilenames: 1.1.
ret: 1.2, 1.3.
ret2: 1.2, 1.3.
rf_common: 1.2.
sc_common: 1.2.
sc_exit: 1.2.
sc_exit_check: 1.2.
sc_neg: 1.2.
sc_pos: 1.2.
sc_target: 1.2.
sc_target_check: 1.2.
sc_wall: 1.2.
sc_wall_check: 1.2.
sector: 1.2.
sector_points: 1.2.
sector_type_pointer: 1.2.
shapecounts: 1.2.
shapysize: 1.2.
shapetype: 1.2.
silo_file: 1.2.
silo_format: 1.2.
silo_precision: 1.2, 1.3.
so_type_num: 1.2.
sp: 1.2.
SP: 1.2.
sp_common: 1.2.
sp_sy: 1.2.
sp_sy_len: 1.2.
st_decls: 1.1, 1.2, 1.3.
status: 1.1, 1.2.

stderr: 1.2.
string_length: 1.2, 1.3.
string_lookup: 1.2.
sy_decls: 1.1, 1.2.
tally_name: 1.2.
test: 1.2.
test_vec_1: 1.2.
test_vec_2: 1.2.
test_vec_3: 1.2.
this_node: 1.2.
this_polygon: 1.2.
time_ind: 1.3.
time_slice: 1.3.
tl_check: 1.2.
tl_common: 1.2.
tl_decls: 1.2.
tl_index_max: 1.2.
tl_index_problem_sp: 1.2.
tl_index_test: 1.2.
tl_index_test_author: 1.2.
tl_index_zone: 1.2.
tl_num: 1.2.
tl_tag_length: 1.2.
triangulate_polygon: 1.
triangulate_to_zones: 1.
trim: 1.2, 1.3.
TRUE: 1.2.

ucd_plot: 1, 1.1.
unit: 1.1, 1.2.

var_alloc: 1.2.
var_free: 1.2.
var_name: 1.3.
var_opts: 1.3.
var_realloca: 1.2.
vc_cross: 1.2.
vc_decl: 1.2.
vc_decls: 1.2.
vc_equal: 1.2.
vc_product: 1.2.
vc_set: 1.2.
vname: 1.2.
vtag: 1.2.

write_silo_data: 1.2, 1.3.

yhat: 1.2.

zero: 1.2.
zn_check: 1.2.
zn_common: 1.2.
zn_decls: 1.2.
zn_num: 1, 1.2.

⟨ Functions and Subroutines 1.2, 1.3 ⟩ Used in section 1.1.
⟨ Memory allocation interface 0 ⟩ Used in sections 1.2 and 1.1.

COMMAND LINE: "fweave -f -i! -W[-ykw800 -ytw40000 -j -n/
/u/dstotler/degas2/src/ucd_plot.web".

WEB FILE: "/u/dstotler/degas2/src/ucd_plot.web".

CHANGE FILE: (none).

GLOBAL LANGUAGE: FORTRAN.